Topics in Compiler Optimization

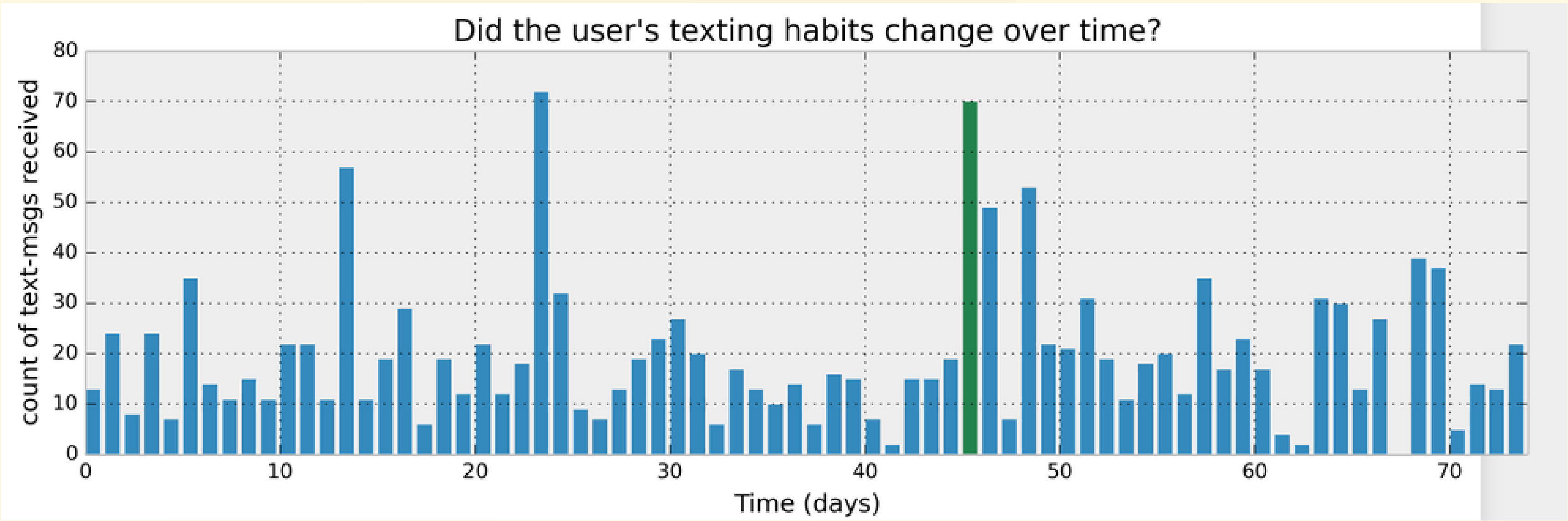
Probabilistic Programming Languages

What is it?

Probabilistic programming is a tool for *statistical modeling*.

Example

Problem



Modeling

Required:

- Discrete distribution for number of texts each day
  Example: `Poisson` distribution
- Continuous distribution over parameters of discrete distribution
- Parameters for the discrete distribution change at some point in time.

For instance, assume that the number of texts each day are λ and

$$\lambda = \begin{cases} \lambda_1 & t < \tau \\ \lambda_2 & t \geq \tau \end{cases}$$

Then, we have

$$\lambda_1 \sim Exp(\alpha)$$

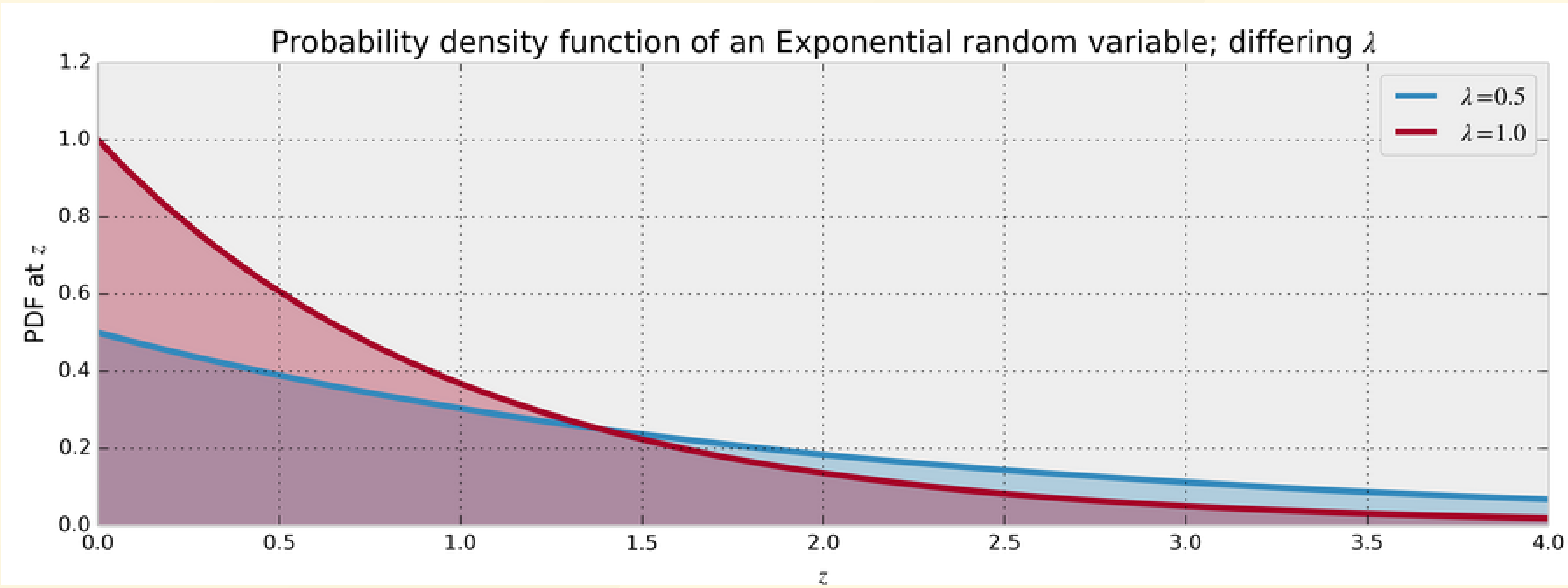
$$\lambda_2 \sim Exp(\alpha)$$

$$\tau \sim \mathrm{DiscreteUniform}(1, 70)$$

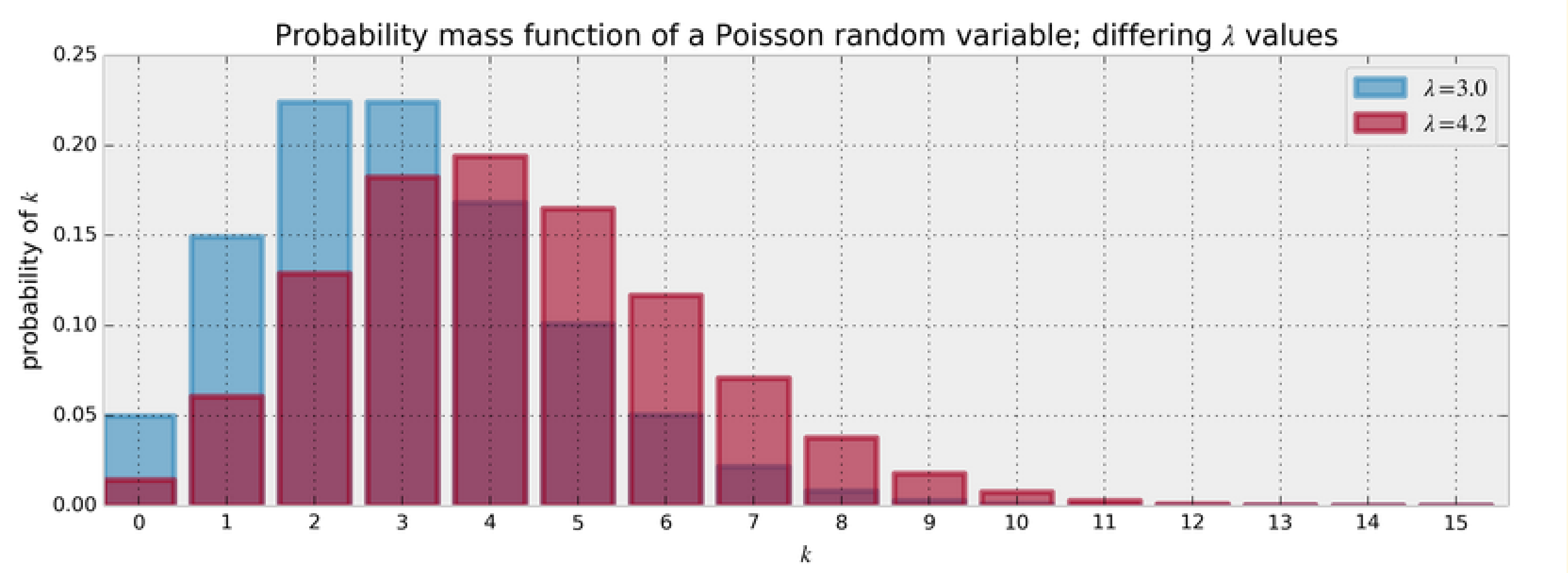
$$C_i \sim \mathrm{Poisson}(\lambda)$$

The parameter α is chosen. Here we take it to be the inverse of the sample mean of the number of messages.

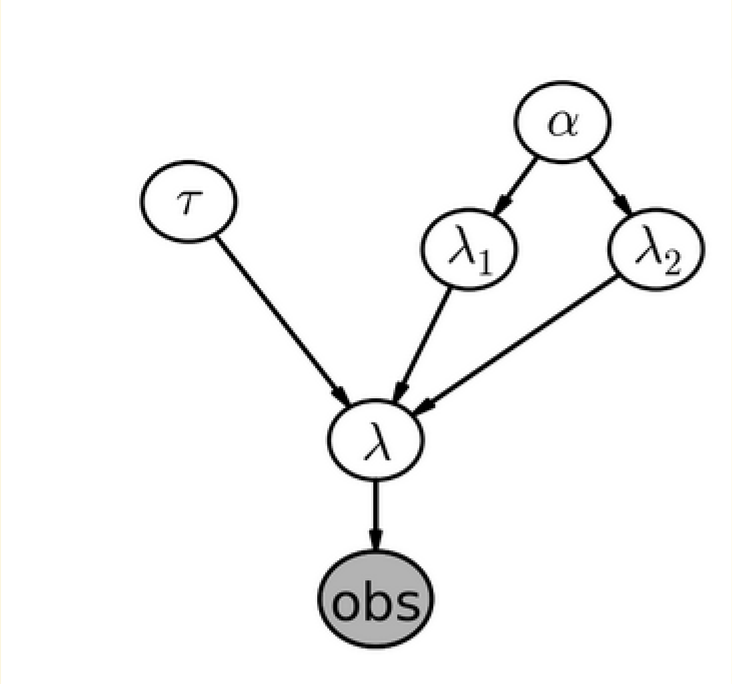
Exponential Distribution



Poission Distribution



Probabilistic Graphical Model



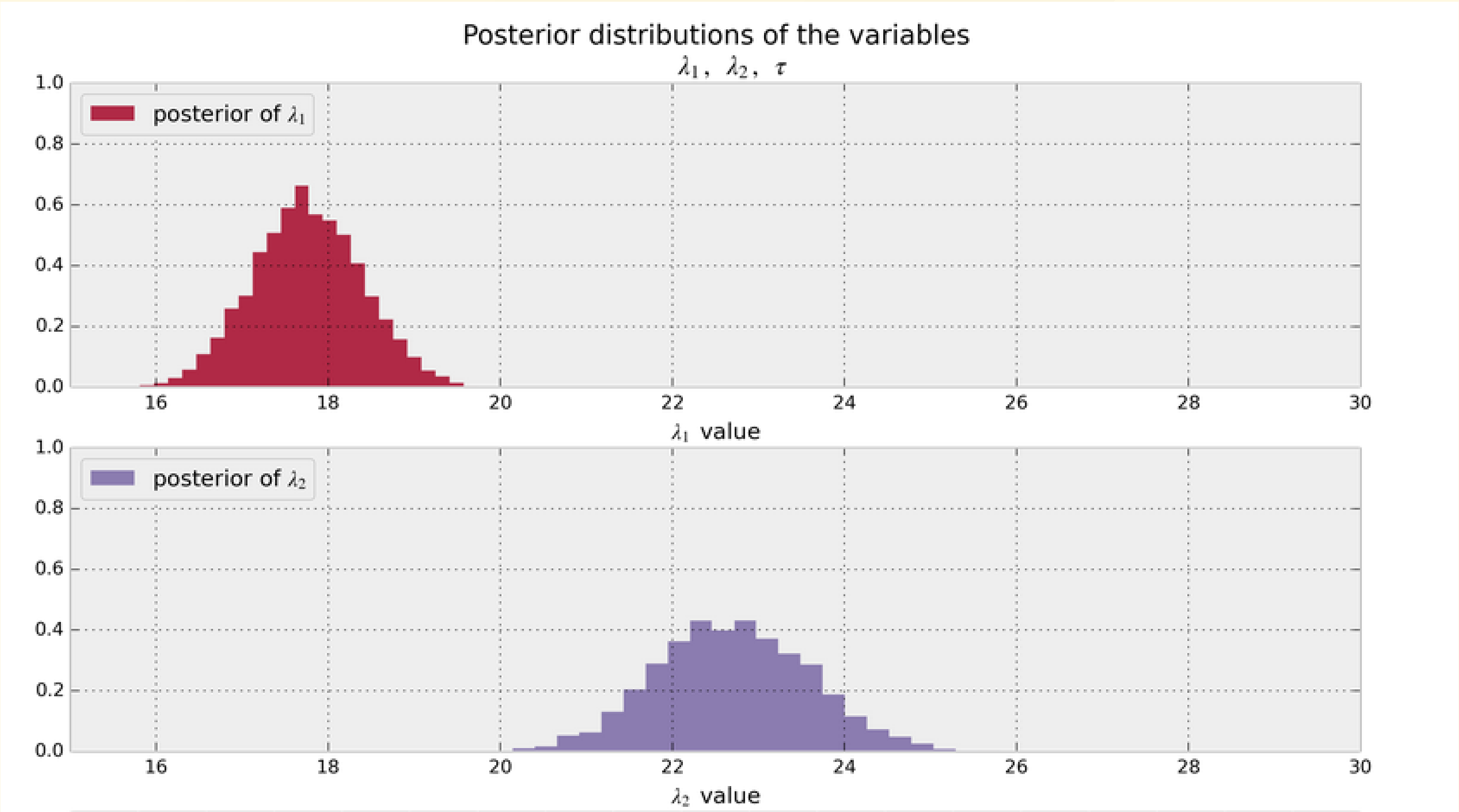
Pseudo Code

```
# count data is given.
observation = poisson("obs", lambda_, value=count_data, observed=True)
model = Model([observation, lambda_1, lambda_2, tau])

# draw samples and infer
mcmc = MCMC(model)
mcmc.sample(40000, 10000, 1)
```

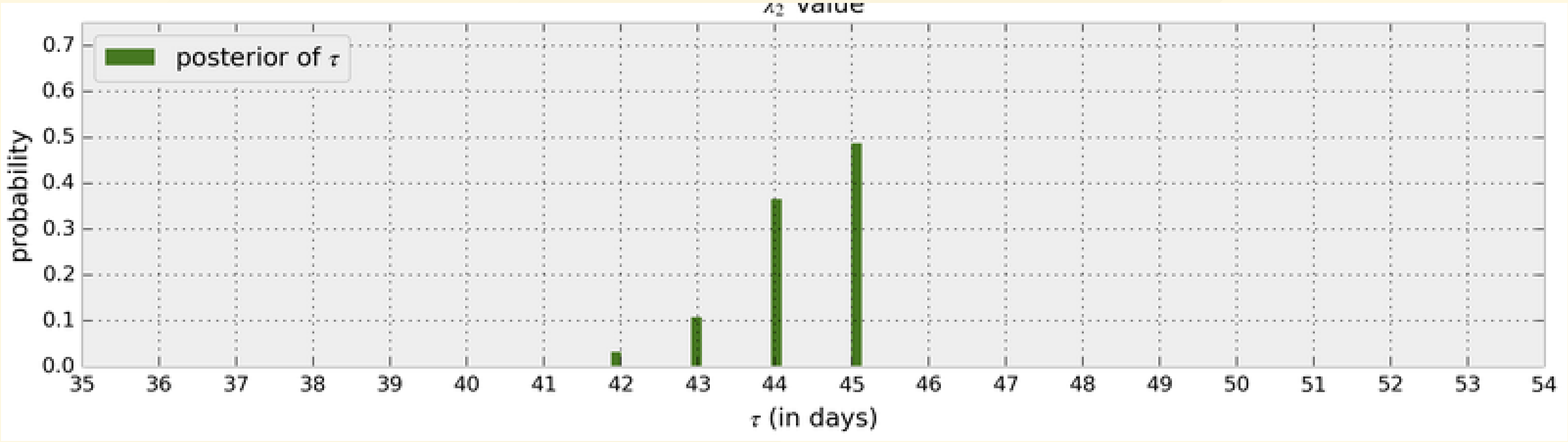
Actual code ~ 20 lines

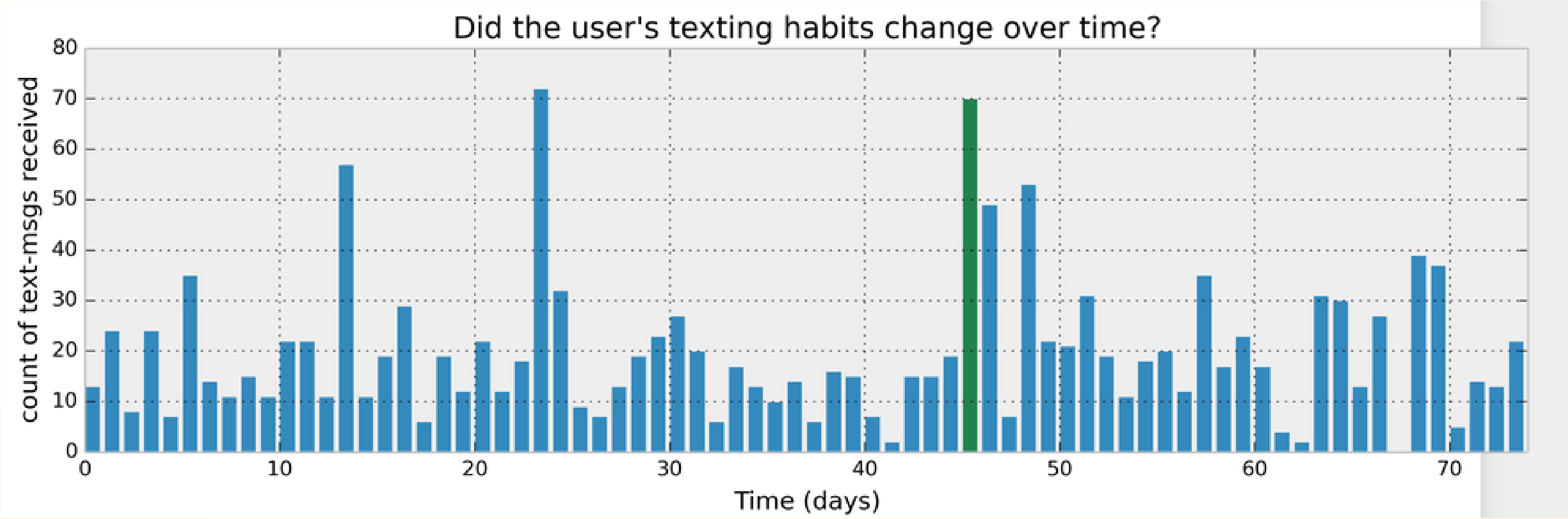
Results



Posterior distributions of the variables
$\lambda_1,\ \lambda_2,\ \tau$
posterior of λ_1
1.0
0.8
0.6
0.4
0.2
0.0
16
18
20
22
24
26
28
30
λ_1 value
posterior of λ_2
1.0
0.8
0.6
0.4
0.2
0.0
16
18
20
22
24
26
28
30
λ_2 value

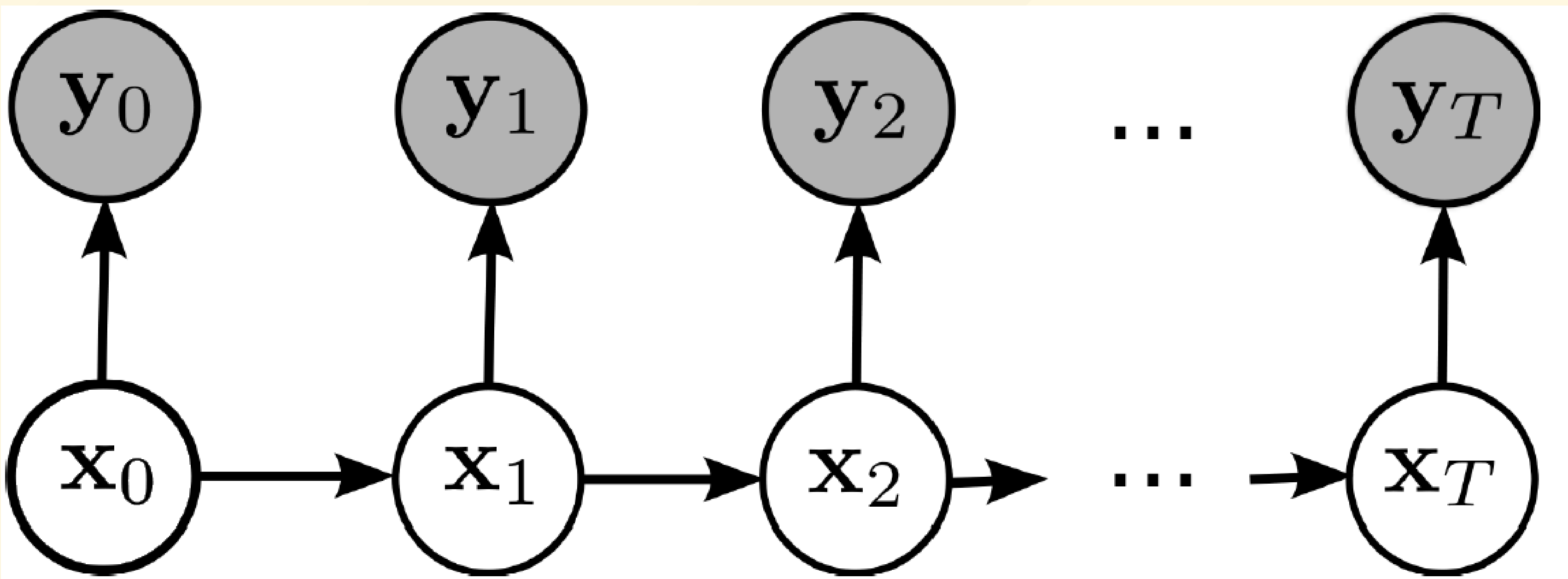
Results (continued)





Optimization Example

Hidden Markov Model



Naive Implementation

```
var transition = function(s) {
  return s ? flip(0.7) : flip(0.3)
}

var observeState = function(s) {
  return s ? flip(0.9) : flip(0.1)
}

observeState(transition(true))

var hmm = function(n) {
  var prev = (n==1) ? {states: [true], observations:[]} : hmm(n-1)
  var newState = transition(prev.states[prev.states.length-1])
  var newObs = observeState(newState)
  return {
    states: prev.states.concat([newState]),
    observations: prev.observations.concat([newObs])
  }
}
hmm(3) # hmm with 3 hidden states

# sample output: # {"states":[true,true,true,true,false],
# "observations":[true,true,true,false]}
```

Factors

```
var binomial = function(){
  var a = sample(Bernoulli({ p: 0.1 }))
  var b = sample(Bernoulli({ p: 0.9 }))
  var c = sample(Bernoulli({ p: 0.1 }))
  factor((a && b) ? 0 : -Infinity)
  return a + b + c
}
```

The `factor` keyword re-weights an execution by adding the given number to the log-probability of that execution.

Decomposing and Interleaving Factors

```
var binomial = function(){
  var a = sample(Bernoulli({ p: 0.1 }))
  factor(a ? 0 : -Infinity)
  var b = sample(Bernoulli({ p: 0.9 }))
  factor(b ? 0 : -Infinity)
  var c = sample(Bernoulli({ p: 0.1 }))
  return a + b + c
}
```

Incrementalizing the HMM

```
var hmmRecur = function(n, states, observations){
  var newState = transition(states[states.length-1])
  var newObs = observeState(newState)
  factor(newObs==trueObs[observations.length] ? 0 : -Infinity)
  var newStates = states.concat([newState])
  var newObservations = observations.concat([newObs])
  return (n==1) ? { states: newStates, observations: newObservations } :
                  hmmRecur(n-1, newStates, newObservations)
}

var hmm = function(n) {
  return hmmRecur(n, [true], [])
}

var model = function(){
  var r = hmm(3)
  return r.states
}
```

Implementation

```
var binomial = function(){
  var a = sampleWithFactor(Bernoulli({ p: 0.1 }),
                           function(v){return v?0:-Infinity})
  var b = sampleWithFactor(Bernoulli({ p: 0.9 }),
                           function(v){return v?0:-Infinity})
  var c = sample(Bernoulli({ p: 0.1 }))
  return a + b + c
}

Infer({ model: binomial, maxExecutions: 2 })
```

Project Topic: Variance Reduction for Black Box Variational Inference

Inference algorithms are key to probabilistic programming languages since performance depends on them.

Variational Inference

Variational inference methods frame a posterior estimation problem as an optimization problem, where the parameters to be optimized adjust a variational "proxy" distribution to be similar to the true posterior.
That is, we wish to estimate $p(z|x)$ with $q(z|\lambda)$.
This is done by maximizing the Evidence Lower BOund (ELBO):

$$\mathcal{L}(\lambda) = E_{q_\lambda}(z)[\log p(x, z) - \log q(z)]$$

Optimization of ELBO

This is done via stochastic updates to the parameters.

$$\lambda_{t+1} = \lambda_t + \rho_t \nabla f(\lambda_t)$$

After some math-fu, we get

$$\nabla_\lambda \mathcal{L}(\lambda) = \frac{1}{S} \sum_{i=1}^{S} [\nabla_\lambda q(z_s|\lambda)](\log p(x, z_s) - \log q(z_s|\lambda))$$

$$z_s \sim q(z|\lambda)$$

The above computes noisy unbiased gradients of the ELBO with Monte Carlo samples from the variational distribution q

Problems with approach

Variance of the gradient estimators can be too large to be useful. In practice, the high variance gradients would require very small steps which would lead to slow convergence. We can reduce this variance using **Rao-Blackwellization**.

Rao-Blackwellization works by replacing the function whose expectation is being approximated by Monte Carlo with another function that has the same expectation but smaller variance.

Rao-Blackwellization

Consider function $J(X, Y)$
We have,

$$\hat{J}(X) := E[J(X, Y)|X]$$

$$E[\hat{J}(X)] = E[J(X, Y)]$$

$$Var(\hat{J}(X)) = Var(J(X, Y)) - E[(J(X, Y) - \hat{J}(X))^2]$$

Then,

$$Var(\hat{J}(X)) \leq Var(J(X, Y))$$

Specifics

Relevant github issue: https://github.com/blei-lab/edward/issues/4

Implementation of **Rao-Blackwellization** for `Edward` which is a probabilistic programming library for probabilistic modeling, inference and criticism.

Milestone #2

Work on *STAN*: probabilistic programming language.
Relevant github issue:
https://github.com/stan-dev/stan/issues/1552
Improvement in ADVI (Automatic Differentiation Variational Inference)

References

Black Box Variational Inference: *Rajesh Ranganath, Sean Gerrish, David M. Blei*
http://www.cs.columbia.edu/~blei/papers/RanganathGerrishBlei2014.pdf

The Design and Implementation of Probabilistic Programming Languages:
Noah D. Goodman and Andreas Stuhlmüller
http://dippl.org/